In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, roc_auc_score, average_precision_score
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
import pandas as pd
import model_get
import pickle as pkl
```

In [2]:

```
import seaborn as sns
from sklearn.metrics import roc_curve, auc, precision_recall_curve

physical_devices = tf.config.experimental.list_physical_devices('GPU')
if len(physical_devices) > 0:
    tf.config.experimental.set_memory_growth(physical_devices[0], False)

def loadGlove(inputpath, outputpath=""):
    data_list = []
    wordEmb = {}
    with open(inputpath) as f:
        for line in f:
            ll = line.strip().split(',')
            ll[0] = str(int(float(ll[0])))
            data_list.append(ll)

            ll_new = [float(i) for i in ll]
            emb = np.array(ll_new[1:], dtype="float32")
            wordEmb[str(int(ll_new[0]))] = emb

    if outputpath != "":
        with open(outputpath) as f:
            for data in data_list:
                f.writelines(' '.join(data))
    return wordEmb


def plotPrecisionRecallCurve(estimators, labels, xtests, ytests, flnm, icol=1):
    colors = sns.color_palette("husl", len(estimators))  # Use a color palette from seaborn
```

```
    indx = 0
    plt.figure(figsize=(10, 6))  # Increase figure size
    plt.plot([0, 1], [0, 1], 'k--')
    for estimator, color in zip(estimators, colors):
        if len(ytests[indx].shape) == 2:
            pre, rec, _ = precision_recall_curve(ytests[indx][:, icol], estimator.predict(xtests[indx])[:, icol], pos_label=icol)
        else:
            pre, rec, _ = precision_recall_curve(ytests[indx], estimator.predict_proba(xtests[indx])[:, icol], pos_label=icol)
        plt.plot(rec, pre, label=labels[indx] + ' (PR AUC: %s \u00B1 0.001)' % (np.round(auc(rec, pre), 3)), color=color)
        indx += 1
    plt.xlabel('Recall', fontsize=14)  # Increase font size
    plt.ylabel('Precision', fontsize=14)  # Increase font size
    plt.legend(loc='best', fontsize=12)  # Increase font size
    plt.savefig(flnm)


def plotRocCurve(estimators, labels, xtests, ytests, flnm, icol=1):
    colors = sns.color_palette("husl", len(estimators))  # Use a color palette from seaborn
    indx = 0
    plt.figure(figsize=(10, 6))  # Increase figure size
    plt.plot([0, 1], [0, 1], 'k--')
    for estimator, color in zip(estimators, colors):
        if len(ytests[indx].shape) == 2:
            fprs, tprs, _ = roc_curve(ytests[indx][:, icol], estimator.predict(xtests[indx])[:, icol])
        else:
            fprs, tprs, _ = roc_curve(ytests[indx], estimator.predict_proba(xtests[indx])[:, icol])
        plt.plot(fprs, tprs, label=labels[indx] + ' (ROC AUC: %s \u00B1 0.001)' % (np.round(auc(fprs, tprs), 3)), color=color)
        indx += 1
    plt.xlabel('False positive rate', fontsize=14)  # Increase font size
    plt.ylabel('True positive rate', fontsize=14)  # Increase font size
    plt.legend(loc='best', fontsize=12)  # Increase font size
    plt.savefig(flnm)
```

In [61]:

```
import random
import os
seed = 123
random.seed(seed)
os.environ['PYTHONHASHSEED']=str(seed)
np.random.seed(seed)
tf.random.set_seed(seed)
```

```
def f1_metric(y_true, y_pred):
    true_positives = tf.keras.backend.sum(tf.keras.backend.round(tf.keras.backend.clip(y_true * y_pred, 0, 1)))
    possible_positives = tf.keras.backend.sum(tf.keras.backend.round(tf.keras.backend.clip(y_true, 0, 1)))
    predicted_positives = tf.keras.backend.sum(tf.keras.backend.round(tf.keras.backend.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + tf.keras.backend.epsilon())
    recall = true_positives / (possible_positives + tf.keras.backend.epsilon())
    f1_val = 2 * (precision * recall) / (precision + recall + tf.keras.backend.epsilon())
    return f1_val


early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_f1_metric', min_delta=0.0001,
    patience=10, verbose=0, mode='max')
callbacks = [early_stopping]

dataset = 'CIRCLE'
num_classes = 2
flpath = 'data/'


retrain = False
```

cross-validation

Crispr_IP

In [62]:

```
# Initialize the cross-validation splitter
cv = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)

print('crispr_ip_model')
encoder_shape = (23, 6)
seg_len, coding_dim = encoder_shape

open_name = 'encoded6x23' + dataset + '.pkl'


print('load data!')
print(open_name)

loaddata = pkl.load(
    open(flpath + open_name, 'rb'),
```

```
    encoding='latin1'
)

# Prepare data for cross-validation
X = np.array(loaddata.images)
y = loaddata.target

eval_fun_names = ['Accuracy', 'F1 score', 'Precision', 'Recall', 'ROC AUC', 'PR AUC']
eval_funs = [accuracy_score, f1_score, precision_score, recall_score, roc_auc_score, average_precision_score]
eval_fun_types = [True, True, True, True, False, False]

# Initialize the result dict
results1 = {name: [] for name in eval_fun_names}

# Load the pre-trained model
model_path = 'saved_model/'+dataset+'crispr_ip.h5'
crispr_ip_model = load_model(model_path, custom_objects={'f1_metric': f1_metric})

# Perform cross-validation
for train_index, test_index in cv.split(X, y):
    X_train_full, X_test = X[train_index], X[test_index]
    y_train_full, y_test = y[train_index], y[test_index]

    # Create a validation set from the full training data
    X_train, X_val, y_train, y_val = train_test_split(
        X_train_full,
        y_train_full,
        stratify=pd.Series(y_train_full),
        test_size=0.2,
        shuffle=True,
        random_state=42)

    # Transform the data
    X_train, X_test1, y_train, y_test1, X_val, y_val, input_shape = model_get.transformIO(
        X_train, X_test, y_train, y_test, X_val, y_val, seg_len, coding_dim, num_classes
    )
    # Calculate metrics
    y_score = crispr_ip_model.predict(X_test1)
    y_pred = np.argmax(y_score, axis=1)
    y_score = y_score[:, 1]
    y_test = np.argmax(y_test1, axis=1)

    # Store the metrics
    for index_f, function in enumerate(eval_funs):
```

```
        if eval_fun_types[index_f]:
            score = np.round(function(y_test, y_pred), 4)
        else:
            score = np.round(function(y_test, y_score), 4)
        results1[eval_fun_names[index_f]].append(score)

# Calculate the average of metrics
avg_results1 = {name: np.mean(scores) for name, scores in results1.items()}

# Print the average metrics
for name, avg_score in avg_results1.items():
    print('{:<15}{:>15}'.format(name, avg_score))
```

```
crispr_ip_model
load data!
encoded6x23CIRCLE.pkl
xtrain shape: (374367, 1, 23, 6)
374367 train samples
116990 test samples
93592 val samples
xtrain shape: (374367, 1, 23, 6)
374367 train samples
116990 test samples
93592 val samples
xtrain shape: (374367, 1, 23, 6)
374367 train samples
116990 test samples
93592 val samples
xtrain shape: (374367, 1, 23, 6)
374367 train samples
116990 test samples
93592 val samples
xtrain shape: (374368, 1, 23, 6)
374368 train samples
116989 test samples
93592 val samples
Accuracy                0.0126
F1 score                0.0249
Precision               0.0126
Recall                     1.0
ROC AUC                0.15006
PR AUC         0.0074399999999999996
```

cnn_std

In [64]:
```
# Initialize the cross-validation splitter
cv = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)

print('cnn_std')
encoder_shape = (23, 4)
seg_len, coding_dim = encoder_shape

open_name = 'encoded4x23' + dataset + '.pkl'


print('load data!')
print(open_name)

loaddata = pkl.load(
    open(flpath + open_name, 'rb'),
    encoding='latin1'
)

# Prepare data for cross-validation
X = np.array(loaddata.images)
y = loaddata.target

eval_fun_names = ['Accuracy', 'F1 score', 'Precision', 'Recall', 'ROC AUC', 'PR AUC']
eval_funs = [accuracy_score, f1_score, precision_score, recall_score, roc_auc_score, average_precision_score]
eval_fun_types = [True, True, True, True, False, False]

# Initialize the result dict
results2 = {name: [] for name in eval_fun_names}

# Load the pre-trained model
model_path = 'saved_model/'+dataset+'cnn_std.h5'
cnn_std_model = load_model(model_path,custom_objects={'f1_metric': f1_metric})

# Perform cross-validation
for train_index, test_index in cv.split(X, y):
    X_train_full, X_test = X[train_index], X[test_index]
    y_train_full, y_test = y[train_index], y[test_index]

    # Create a validation set from the full training data
    X_train, X_val, y_train, y_val = train_test_split(
        X_train_full,
        y_train_full,
```

```
        stratify=pd.Series(y_train_full),
        test_size=0.2,
        shuffle=True,
        random_state=42)

    # Transform the data
    X_train, X_test2, y_train, y_test2, X_val, y_val, input_shape = model_get.cnn_std_transformIO(
        X_train, X_test, y_train, y_test, X_val, y_val, seg_len, coding_dim, num_classes
    )
    # Calculate metrics
    y_score = cnn_std_model.predict(X_test2)
    y_pred = np.argmax(y_score, axis=1)
    y_score = y_score[:, 1]
    y_test = np.argmax(y_test2, axis=1)

    # Store the metrics
    for index_f, function in enumerate(eval_funs):
        if eval_fun_types[index_f]:
            score = np.round(function(y_test, y_pred), 4)
        else:
            score = np.round(function(y_test, y_score), 4)
        results2[eval_fun_names[index_f]].append(score)

# Calculate the average of metrics
avg_results2 = {name: np.mean(scores) for name, scores in results2.items()}

# Print the average metrics
for name, avg_score in avg_results2.items():
    print('{:<15}{:>15}'.format(name, avg_score))
```

```
cnn_std
load data!
encoded4x23CIRCLE.pkl
xtrain shape: (374367, 1, 23, 4)
374367 train samples
116990 test samples
93592 xval samples
xtrain shape: (374367, 1, 23, 4)
374367 train samples
116990 test samples
93592 xval samples
xtrain shape: (374367, 1, 23, 4)
374367 train samples
116990 test samples
93592 xval samples
```

```
xtrain shape: (374367, 1, 23, 4)
374367 train samples
116990 test samples
93592 xval samples
xtrain shape: (374368, 1, 23, 4)
374368 train samples
116989 test samples
93592 xval samples
Accuracy       0.9480999999999999
F1 score               0.04994
Precision              0.03252
Recall                  0.1081
ROC AUC                0.50236
PR AUC                 0.02194
```

crisprDNT

In [65]:

```
from model_get import PositionalEncoding
from keras_multi_head import MultiHeadAttention

# Initialize the cross-validation splitter
cv = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)

print('crisprDNT')
encoder_shape = (23, 14)
seg_len, coding_dim = encoder_shape

open_name = 'encodedmismatchtype14x23' + dataset + '.pkl'


print('load data!')
print(open_name)

loaddata = pkl.load(
    open(flpath + open_name, 'rb'),
    encoding='latin1'
)

# Prepare data for cross-validation
X = np.array(loaddata.images)
y = loaddata.target

eval_fun_names = ['Accuracy', 'F1 score', 'Precision', 'Recall', 'ROC AUC', 'PR AUC']
eval_funs = [accuracy_score, f1_score, precision_score, recall_score, roc_auc_score, average_precision_score]
```

```
eval_fun_types = [True, True, True, True, False, False]

# Initialize the result dict
results3 = {name: [] for name in eval_fun_names}

# Load the pre-trained model
model_path = 'saved_model/'+dataset+'crisprDNT.h5'
crisprDNT = load_model(model_path, custom_objects={'PositionalEncoding': PositionalEncoding,'MultiHeadAttention':MultiHeadAttention,'

# Perform cross-validation
for train_index, test_index in cv.split(X, y):
    X_train_full, X_test = X[train_index], X[test_index]
    y_train_full, y_test = y[train_index], y[test_index]

    # Create a validation set from the full training data
    X_train, X_val, y_train, y_val = train_test_split(
        X_train_full,
        y_train_full,
        stratify=pd.Series(y_train_full),
        test_size=0.2,
        shuffle=True,
        random_state=42)

    # Transform the data
    X_train, X_test3, y_train, y_test3, X_val, y_val, input_shape = model_get.transformIO(
        X_train, X_test, y_train, y_test, X_val, y_val, seg_len, coding_dim, num_classes
    )
    # Calculate metrics
    y_score = crisprDNT.predict(X_test3)
    y_pred = np.argmax(y_score, axis=1)
    y_score = y_score[:, 1]
    y_test = np.argmax(y_test3, axis=1)

    # Store the metrics
    for index_f, function in enumerate(eval_funs):
        if eval_fun_types[index_f]:
            score = np.round(function(y_test, y_pred), 4)
        else:
            score = np.round(function(y_test, y_score), 4)
        results3[eval_fun_names[index_f]].append(score)

# Calculate the average of metrics
avg_results3 = {name: np.mean(scores) for name, scores in results3.items()}
```

```
# Print the average metrics
for name, avg_score in avg_results3.items():
    print('{:<15}{:>15}'.format(name, avg_score))
```

```
crisprDNT
load data!
encodedmismatchtype14x23CIRCLE.pkl
WARNING:tensorflow:Layer lstm_27 will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will use generic GPU k
ernel as fallback when running on GPU
WARNING:tensorflow:Layer lstm_27 will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will use generic GPU k
ernel as fallback when running on GPU
WARNING:tensorflow:Layer lstm_27 will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will use generic GPU k
ernel as fallback when running on GPU
xtrain shape: (374367, 1, 23, 14)
374367 train samples
116990 test samples
93592 val samples
xtrain shape: (374367, 1, 23, 14)
374367 train samples
116990 test samples
93592 val samples
xtrain shape: (374367, 1, 23, 14)
374367 train samples
116990 test samples
93592 val samples
xtrain shape: (374367, 1, 23, 14)
374367 train samples
116990 test samples
93592 val samples
xtrain shape: (374368, 1, 23, 14)
374368 train samples
116989 test samples
93592 val samples
Accuracy               0.99606
F1 score               0.84486
Precision      0.8313200000000001
Recall                  0.8589
ROC AUC        0.9886199999999998
PR AUC         0.8814599999999999
```

crisprNet

In [66]:

```
# Initialize the cross-validation splitter
cv = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)

print('CRISPR_Net')
```

```
encoder_shape = (23, 7)
seg_len, coding_dim = encoder_shape

open_name = 'encoded7x23' + dataset + '.pkl'


print('load data!')
print(open_name)

loaddata = pkl.load(
    open(flpath + open_name, 'rb'),
    encoding='latin1'
)

# Prepare data for cross-validation
X = np.array(loaddata.images)
y = loaddata.target

eval_fun_names = ['Accuracy', 'F1 score', 'Precision', 'Recall', 'ROC AUC', 'PR AUC']
eval_funs = [accuracy_score, f1_score, precision_score, recall_score, roc_auc_score, average_precision_score]
eval_fun_types = [True, True, True, True, False, False]

# Initialize the result dict
results4 = {name: [] for name in eval_fun_names}

# Load the pre-trained model
model_path = 'saved_model/'+dataset+'CRISPR_Net.h5'
CRISPR_Net_model = load_model(model_path, custom_objects={'f1_metric': f1_metric})

# Perform cross-validation
for train_index, test_index in cv.split(X, y):
    X_train_full, X_test = X[train_index], X[test_index]
    y_train_full, y_test = y[train_index], y[test_index]

    # Create a validation set from the full training data
    X_train, X_val, y_train, y_val = train_test_split(
        X_train_full,
        y_train_full,
        stratify=pd.Series(y_train_full),
        test_size=0.2,
        shuffle=True,
        random_state=42)

    # Transform the data
```

```
    X_train, X_test4, y_train, y_test4, X_val, y_val, input_shape = model_get.CRISPR_Net_transformIO(
        X_train, X_test, y_train, y_test, X_val, y_val, seg_len, coding_dim, num_classes
    )
    # Calculate metrics
    y_score = CRISPR_Net_model.predict(X_test4)
    y_pred = np.argmax(y_score, axis=1)
    y_score = y_score[:, 1]
    y_test = np.argmax(y_test4, axis=1)

    # Store the metrics
    for index_f, function in enumerate(eval_funs):
        if eval_fun_types[index_f]:
            score = np.round(function(y_test, y_pred), 4)
        else:
            score = np.round(function(y_test, y_score), 4)
        results4[eval_fun_names[index_f]].append(score)

# Calculate the average of metrics
avg_results4 = {name: np.mean(scores) for name, scores in results4.items()}

# Print the average metrics
for name, avg_score in avg_results4.items():
    print('{:<15}{:>15}'.format(name, avg_score))
```

```
CRISPR_Net
load data!
encoded7x23CIRCLE.pkl
xtrain shape: (374367, 1, 23, 7)
374367 train samples
116990 test samples
93592 val samples
xtrain shape: (374367, 1, 23, 7)
374367 train samples
116990 test samples
93592 val samples
xtrain shape: (374367, 1, 23, 7)
374367 train samples
116990 test samples
93592 val samples
xtrain shape: (374367, 1, 23, 7)
374367 train samples
116990 test samples
93592 val samples
xtrain shape: (374368, 1, 23, 7)
374368 train samples
116989 test samples
```

```
93592 val samples
Accuracy                0.9898
F1 score               0.69384
Precision      0.5589000000000001
Recall                 0.91494
ROC AUC        0.9925599999999999
PR AUC                 0.81404
```

cnnCRISPR

In [58]:

```
# Initialize the cross-validation splitter
cv = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)

print('cnn_crispr model')
print("GloVe model loaded")
VOCAB_SIZE = 16  # 4**3
EMBED_SIZE = 100
glove_inputpath = "data/keras_GloVeVec_" + dataset + "_5_100_10000.csv"
# load GloVe model
model_glove = loadGlove(glove_inputpath)
embedding_weights = np.zeros((VOCAB_SIZE, EMBED_SIZE))
for i in range(VOCAB_SIZE):
    embedding_weights[i, :] = model_glove[str(i)]

open_name = 'encoded_CnnCrispr_' + dataset + '.pkl'

print('load data!')
print(open_name)

loaddata = pkl.load(
    open(flpath + open_name, 'rb'),
    encoding='latin1'
)

# Prepare data for cross-validation
X = np.array(loaddata.images)
y = loaddata.target

eval_fun_names = ['Accuracy', 'F1 score', 'Precision', 'Recall', 'ROC AUC', 'PR AUC']
eval_funs = [accuracy_score, f1_score, precision_score, recall_score, roc_auc_score, average_precision_score]
eval_fun_types = [True, True, True, True, False, False]

# Initialize the result dict
results5 = {name: [] for name in eval_fun_names}
```

```
# Load the pre-trained model
model_path = 'saved_model/'+dataset+'CnnCrispr.h5'
CnnCrispr_model = load_model(model_path, custom_objects={'f1_metric': f1_metric})

# Perform cross-validation
for train_index, test_index in cv.split(X, y):
    X_train_full, X_test = X[train_index], X[test_index]
    y_train_full, y_test = y[train_index], y[test_index]

    # Create a validation set from the full training data
    X_train, X_val, y_train, y_val = train_test_split(
        X_train_full,
        y_train_full,
        stratify=pd.Series(y_train_full),
        test_size=0.2,
        shuffle=True,
        random_state=42)

    # Transform the data
    X_train, X_test5, y_train, y_test5, X_val, y_val = model_get.offt_transformIO(
        X_train, X_test, y_train, y_test, X_val, y_val, num_classes
    )
    # Calculate metrics
    y_score = CnnCrispr_model.predict(X_test5)
    y_pred = np.argmax(y_score, axis=1)
    y_score = y_score[:, 1]
    y_test = np.argmax(y_test5, axis=1)

    # Store the metrics
    for index_f, function in enumerate(eval_funs):
        if eval_fun_types[index_f]:
            score = np.round(function(y_test, y_pred), 4)
        else:
            score = np.round(function(y_test, y_score), 4)
        results5[eval_fun_names[index_f]].append(score)

# Calculate the average of metrics
avg_results5 = {name: np.mean(scores) for name, scores in results5.items()}

# Print the average metrics
for name, avg_score in avg_results5.items():
    print('{:<15}{:>15}'.format(name, avg_score))
```

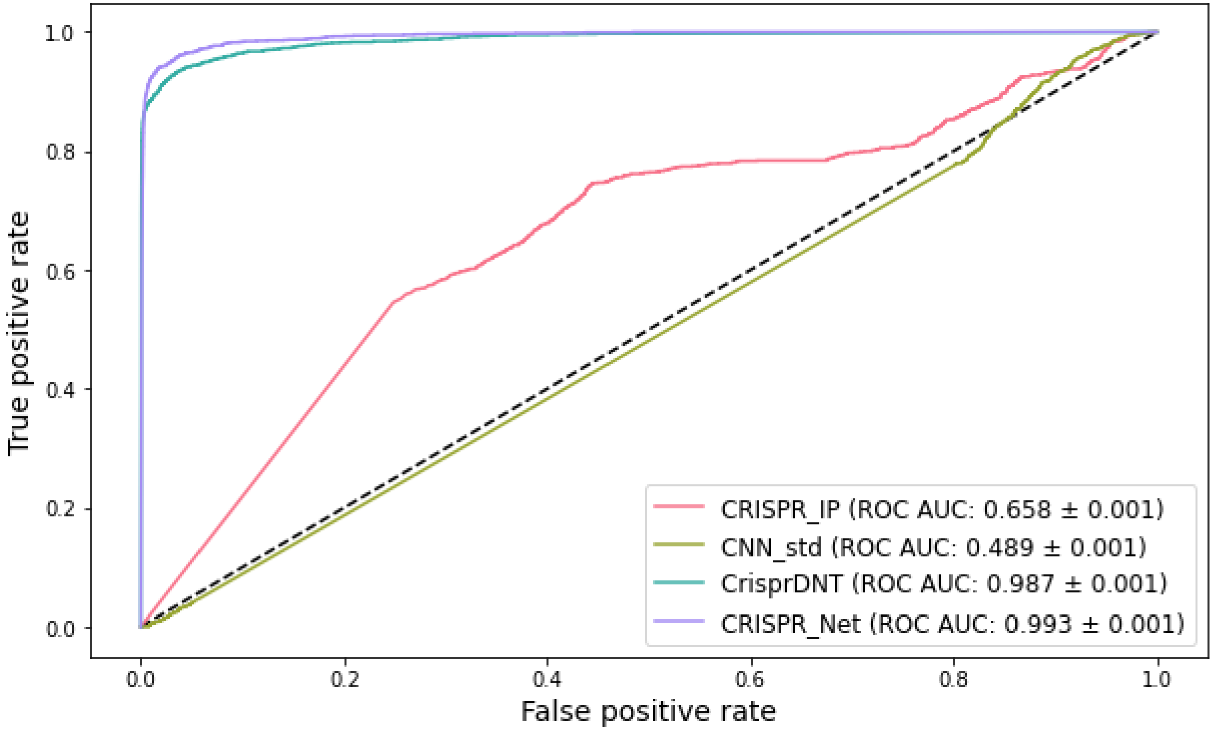
```
cnn_crispr model
GloVe model loaded
load data!
encoded_CnnCrispr_k562.pkl
xtrain shape: (11797, 23)
11797 train samples
3687 test samples
2950 val samples
xtrain shape: (11797, 23)
11797 train samples
3687 test samples
2950 val samples
xtrain shape: (11797, 23)
11797 train samples
3687 test samples
2950 val samples
xtrain shape: (11797, 23)
11797 train samples
3687 test samples
2950 val samples
xtrain shape: (11798, 23)
11798 train samples
3686 test samples
2950 val samples
Accuracy	0.20657999999999999
F1 score		0.01202
Precision		0.00606
Recall		0.7546999999999999
ROC AUC		0.51092
PR AUC		0.00726
```

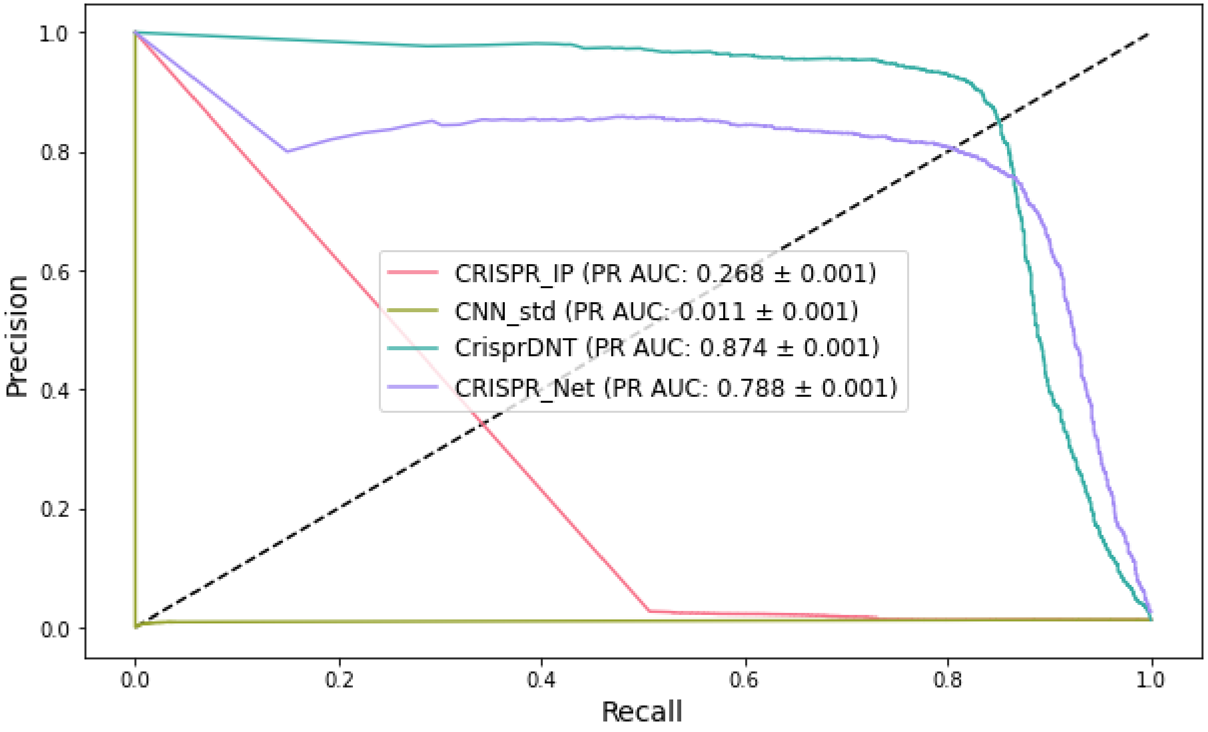
Plot

In [67]:

```
# models = [ crispr_ip_model, cnn_std_model, crisprDNT, CRISPR_Net_model, CnnCrispr_model]

# labels = ['CRISPR_IP', 'CNN_std', 'CrisprDNT', 'CRISPR_Net', 'CnnCrispr']

# xtests = [X_test1, X_test2, X_test3, X_test4, X_test5]

# ytests = [y_test1, y_test2, y_test3, y_test4, y_test5]

# roc_name = 'fig/ROC_5fold_' + dataset + '.png'
# pr_name = 'fig/PR_5fold_' + dataset + '.png'
```

```
# plotRocCurve(models, labels, xtests, ytests, roc_name)

# plotPrecisionRecallCurve(models, labels, xtests, ytests, pr_name)
models = [ crispr_ip_model, cnn_std_model, crisprDNT, CRISPR_Net_model]

labels = ['CRISPR_IP', 'CNN_std', 'CrisprDNT', 'CRISPR_Net']

xtests = [X_test1, X_test2, X_test3, X_test4]

ytests = [y_test1, y_test2, y_test3, y_test4]

roc_name = 'fig/ROC_5fold_' + dataset + '.png'
pr_name = 'fig/PR_5fold_' + dataset + '.png'

plotRocCurve(models, labels, xtests, ytests, roc_name)

plotPrecisionRecallCurve(models, labels, xtests, ytests, pr_name)
```





In [32]:

```
def plot_metrics(metrics, labels, filename):
    """
    :param metrics: dict, keys are metric names, values are lists of metric scores for each model
    :param labels: list of str, model names
    :param filename: str, the name of the file where the plot will be saved
    """
    num_metrics = len(metrics.keys())
    fig, axes = plt.subplots(num_metrics, 1, figsize=(10, num_metrics * 5))

    # Create a color palette
    colors = sns.color_palette("husl", len(labels))

    for ax, metric in zip(axes, metrics.keys()):
        bars = ax.bar(labels, metrics[metric], color=colors)
        ax.set_title(metric, fontsize=16)
        ax.set_ylim([0, 1.05])  # Adjust the y-axis range to allow space for text above bars
        ax.set_ylabel(metric, fontsize=14)

        # Add data values on top of each bar
```

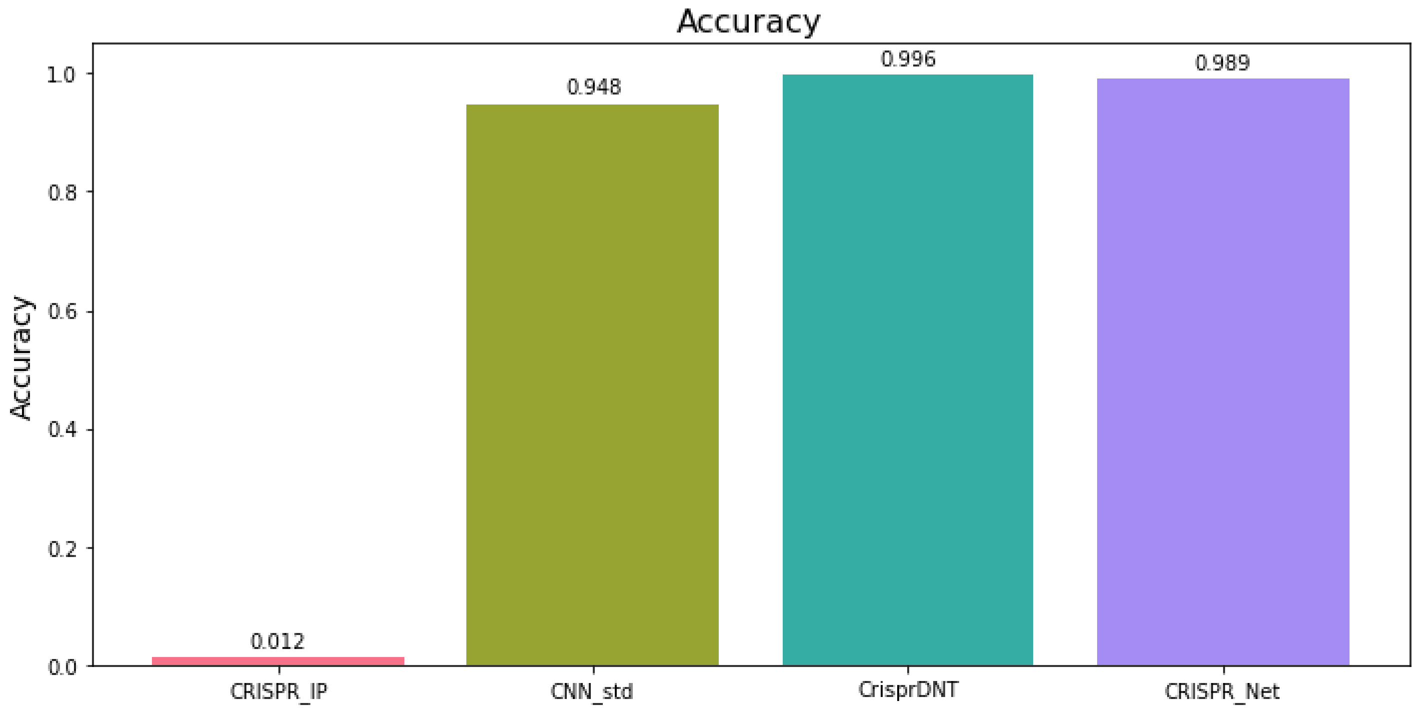
```
        for bar in bars:
            height = bar.get_height()
            value = int(height * 1000) / 1000.0  # Truncate to three decimal places
            ax.text(bar.get_x() + bar.get_width() / 2, height + 0.01, '{:.3f}'.format(value),
                    ha='center', va='bottom', fontsize=10)

    # Save the figure
    plt.tight_layout()
    plt.savefig(filename)
```

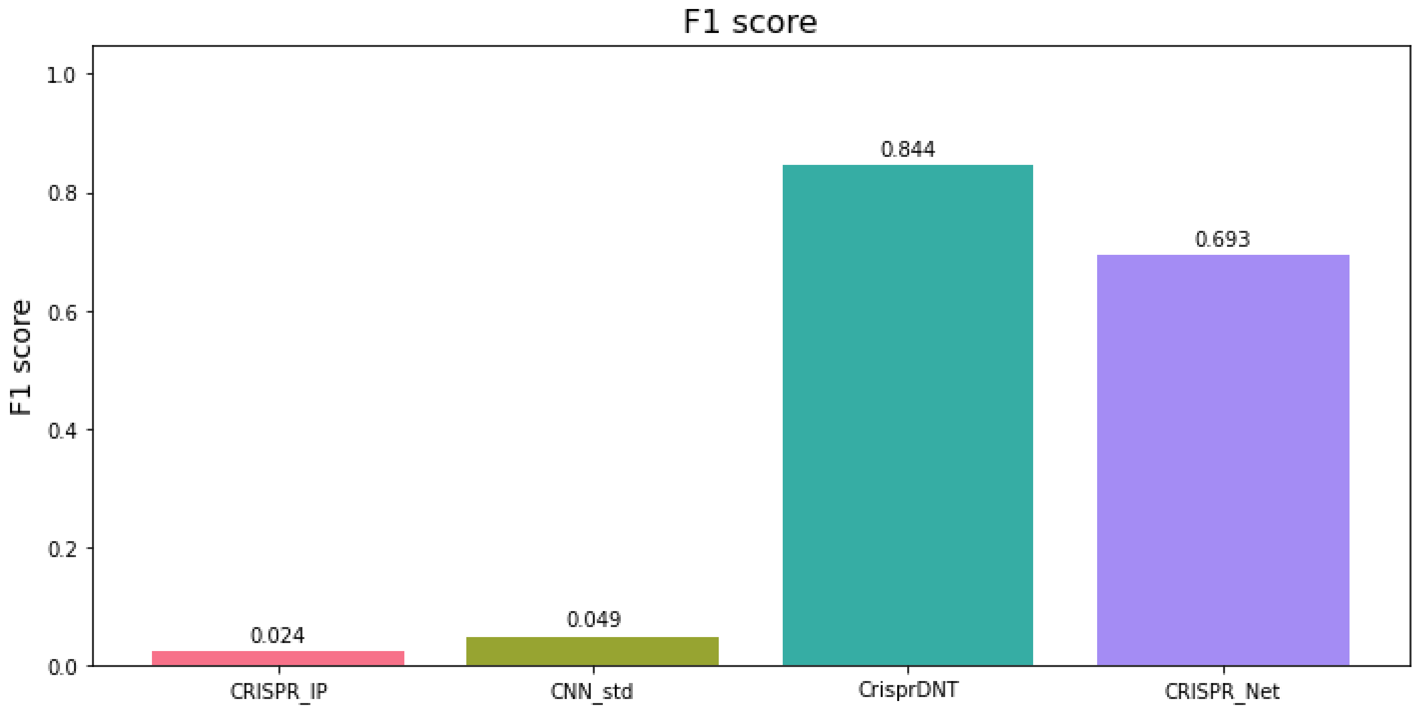
In [70]:

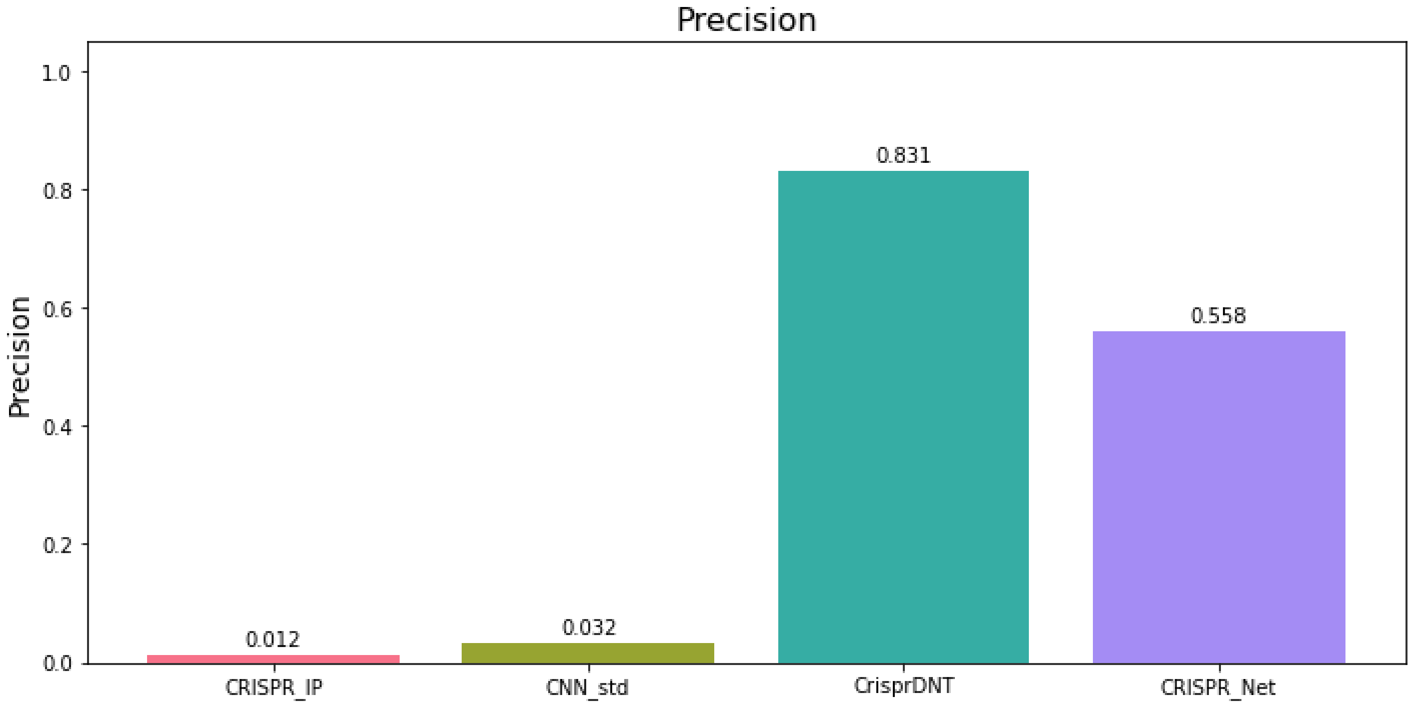
```
# # Extract the metrics for each model
# accuracy_scores1 = avg_results1['Accuracy']
# accuracy_scores2 = avg_results2['Accuracy']
# accuracy_scores3 = avg_results3['Accuracy']
# accuracy_scores4 = avg_results4['Accuracy']
# accuracy_scores5 = avg_results5['Accuracy']

# f1_scores1 = avg_results1['F1 score']
# f1_scores2 = avg_results2['F1 score']
# f1_scores3 = avg_results3['F1 score']
# f1_scores4 = avg_results4['F1 score']
# f1_scores5 = avg_results5['F1 score']

# precision_scores1 = avg_results1['Precision']
# precision_scores2 = avg_results2['Precision']
# precision_scores3 = avg_results3['Precision']
# precision_scores4 = avg_results4['Precision']
# precision_scores5 = avg_results5['Precision']

# recall_scores1 = avg_results1['Recall']
# recall_scores2 = avg_results2['Recall']
# recall_scores3 = avg_results3['Recall']
# recall_scores4 = avg_results4['Recall']
# recall_scores5 = avg_results5['Recall']

# # Create dictionaries for each metric that contains the scores for all models
# accuracy_scores = [accuracy_scores1, accuracy_scores2, accuracy_scores3, accuracy_scores4, accuracy_scores5]
# f1_scores = [f1_scores1, f1_scores2, f1_scores3, f1_scores4, f1_scores5]
# precision_scores = [precision_scores1, precision_scores2, precision_scores3, precision_scores4, precision_scores5]
# recall_scores = [recall_scores1, recall_scores2, recall_scores3, recall_scores4, recall_scores5]

# metrics = {
```

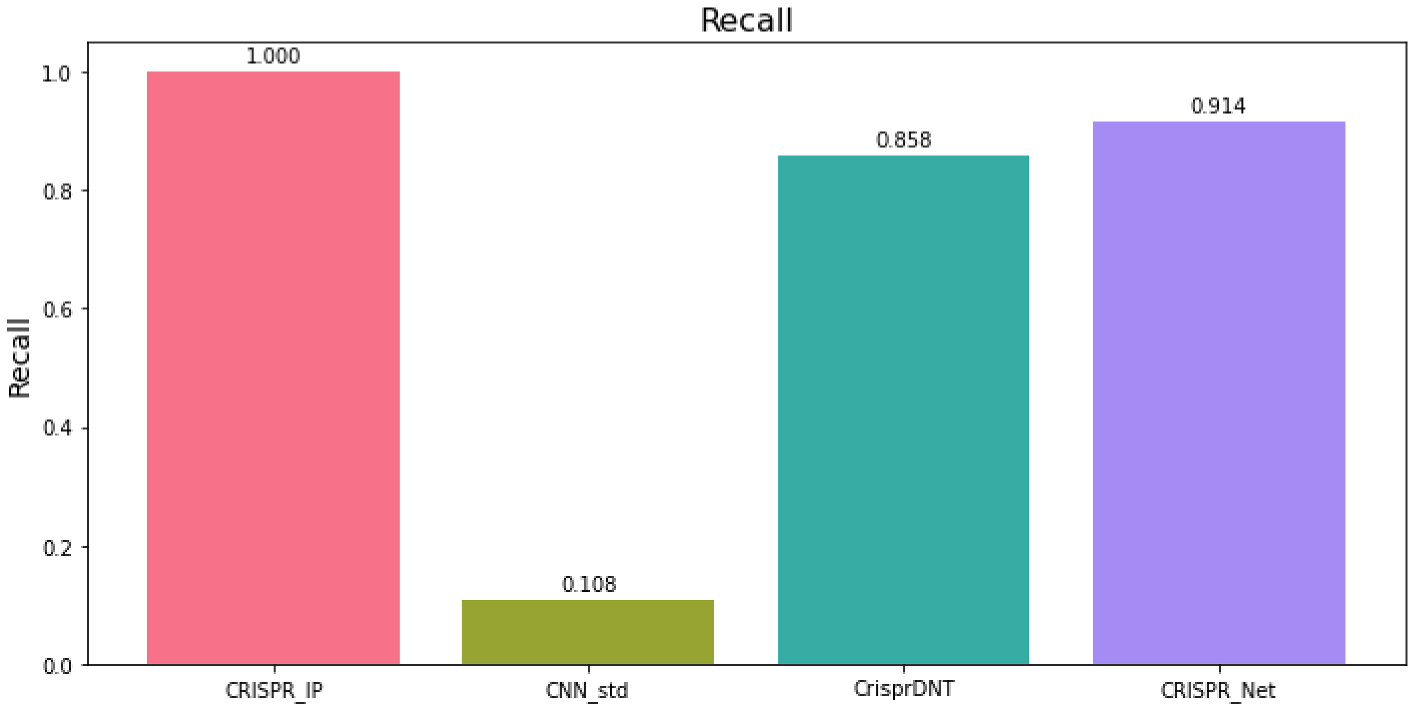
```
#     'Accuracy': accuracy_scores,
#     'F1 score': f1_scores,
#     'Precision': precision_scores,
#     'Recall': recall_scores
# }


# labels = ['CRISPR_IP', 'CNN_std', 'CrisprDNT', 'CRISPR_Net', 'CnnCrispr']

# plot_metrics(metrics, labels, 'fig/5-fold_metrics_plot_hek293t.png')
# Extract the metrics for each model
accuracy_scores1 = avg_results1['Accuracy']
accuracy_scores2 = avg_results2['Accuracy']
accuracy_scores3 = avg_results3['Accuracy']
accuracy_scores4 = avg_results4['Accuracy']


f1_scores1 = avg_results1['F1 score']
f1_scores2 = avg_results2['F1 score']
f1_scores3 = avg_results3['F1 score']
f1_scores4 = avg_results4['F1 score']

precision_scores1 = avg_results1['Precision']
precision_scores2 = avg_results2['Precision']
precision_scores3 = avg_results3['Precision']
precision_scores4 = avg_results4['Precision']


recall_scores1 = avg_results1['Recall']
recall_scores2 = avg_results2['Recall']
recall_scores3 = avg_results3['Recall']
recall_scores4 = avg_results4['Recall']


# Create dictionaries for each metric that contains the scores for all models
accuracy_scores = [accuracy_scores1, accuracy_scores2, accuracy_scores3, accuracy_scores4]
f1_scores = [f1_scores1, f1_scores2, f1_scores3, f1_scores4]
precision_scores = [precision_scores1, precision_scores2, precision_scores3, precision_scores4]
recall_scores = [recall_scores1, recall_scores2, recall_scores3, recall_scores4]

metrics = {
    'Accuracy': accuracy_scores,
    'F1 score': f1_scores,
    'Precision': precision_scores,
```

```
    'Recall': recall_scores
}


labels = ['CRISPR_IP', 'CNN_std', 'CrisprDNT', 'CRISPR_Net']

plot_metrics(metrics, labels, 'fig/5-fold_metrics_plot_CICLE.png')
```


Accuracy
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0.012
0.948
0.996
0.989
CRISPR_IP
CNN_std
CrisprDNT
CRISPR_Net


F1 score
F1 score
1.0
0.8
0.6
0.4
0.2
0.0
0.024
0.049
0.844
0.693
CRISPR_IP
CNN_std
CrisprDNT
CRISPR_Net


Precision
Precision
1.0
0.8
0.6
0.4
0.2
0.0
0.012
0.032
0.831
0.558
CRISPR_IP
CNN_std
CrisprDNT
CRISPR_Net


Recall
Recall
1.0
0.8
0.6
0.4
0.2
0.0
1.000
0.108
0.858
0.914
CRISPR_IP
CNN_std
CrisprDNT
CRISPR_Net

In [ ]: